



嵌入式系统

Embedded System

毛维杰

杭州 • 浙江大学 • 2021

第五章 中断系统

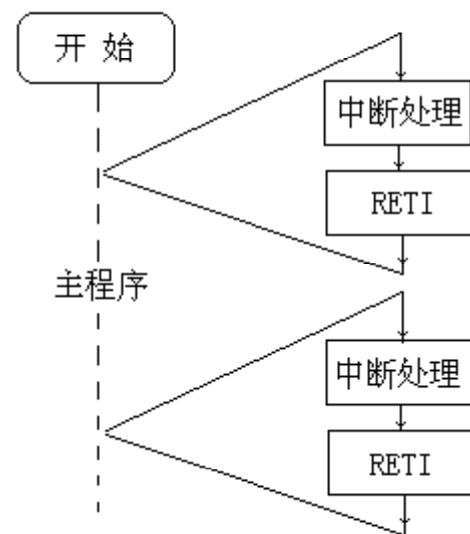
- § 5-1 中断基本概念
- § 5-2 MCS-51中断系统
- § 5-3 中断应用编程

§ 5.1 中断的基本概念

计算机与外设交换信息时，存在高速**CPU**与低速外设之间的矛盾。主要传送方式：

- (1) 无条件传送方式：**外设对计算机来说总是准备好的。**
- (2) 查询传送方式：**传送前计算机先查询外设的状态，若已经准备好就传送，否则就继续查询/等待。**
- (3) 中断传送方式：**外设通过申请中断方式与计算机进行数据传送。**
- (4) 直接存储器存取方式(DMA)：**传送数据的双方直接通过总线传送数据，不经CPU中转。**

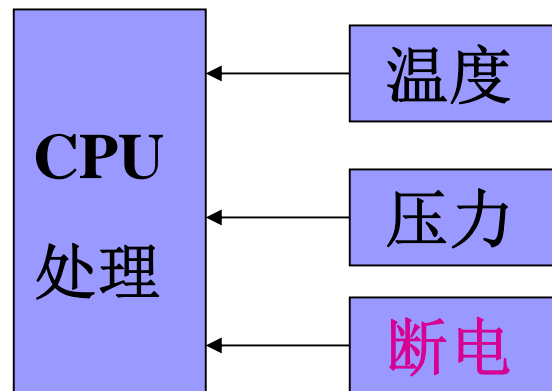
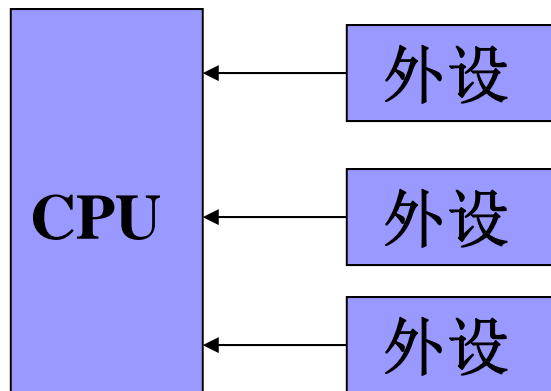
中断



定义：CPU放下正在执行的程序，去处理临时发生的事情，处理完毕后，再返回去继续执行暂停的程序。

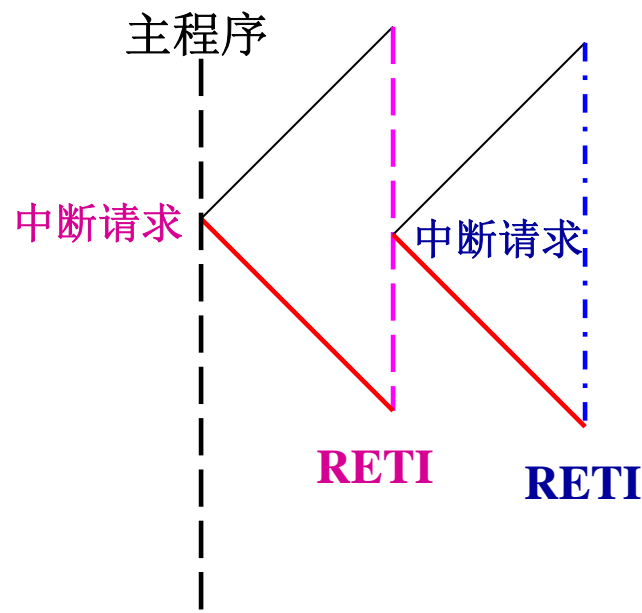
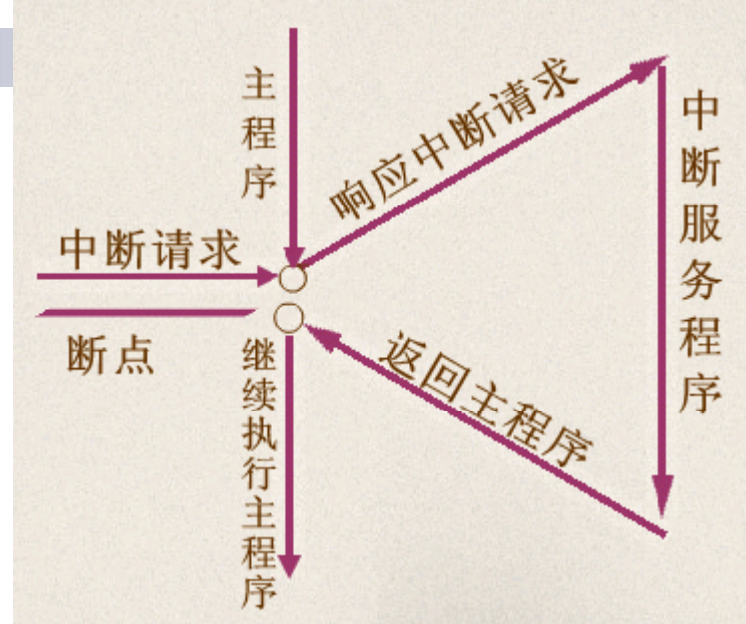
中断的优点

- 实现实时处理
- 执行效果高
- 故障处理



中断系统功能

- 中断的响应和返回
 - 能够根据需要用指令控制中断的开放和关闭
- 中断的分级和优先权
- 中断的嵌套



中断涉及的概念

- 主程序：原来运行的程序；
- 断点：主程序被中断的位置；
- 中断源：引起中断的原因或来源；
- 中断请求：要求中断服务的请求；
- 中断服务程序：中断后执行的处理程序。
- 调用中断服务程序与调用子程序的差别
 - 调用中断服务程序的过程由硬件自动完成

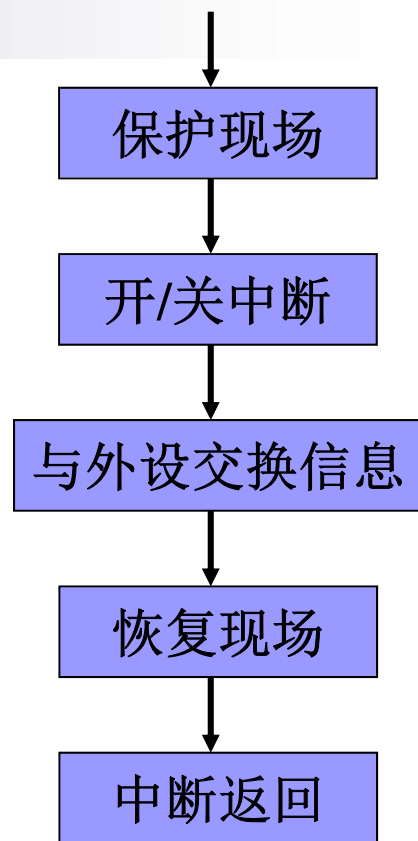
中断服务

中断服务子程序是由用户根据自己的需要编制的，编制时要注意如下问题：

- **1. 保护现场：**由一系列的**PUSH**指令完成。
目的是为了那些与主程序中有冲突的寄存器，(如**A**，**PSW**，工作寄存器等)，如果中断服务子程序中所使用的寄存器与主程序中所使用的寄存器等没有冲突的话，这一步骤可以省略。
- **2. 开/关中断：**由**SETB**或**CLR**指令实现。目的是为了控制中断。

中断服务(2)

- **3. 中断服务：**与中断源实现信息交换
- **4. 恢复现场：**由一系列的**POP**指令完成。是与保护现场对应的，但要注意数据恢复的次序，以免混乱。
- **5. 返回：**使用中断返回指令**RETI**。不能使用一般的子程序返回指令**RET**，因为**RETI**指令除了能恢复断点地址外，还能恢复中断响应时的标志寄存器的值，而这后一个动作是**RET**指令不能完成的。



中断处理原则

1. 当多个中断源同时提出申请时，**CPU**按优先级的高低（**硬件**）由高到低依次为外设服务。**同级按照查询顺序（软件）**
2. 当**CPU**正为某外设服务又有新外设提出申请时，
 - (1) 若新外设级别高则**CPU**终止为原外设服务转去为新外设服务，为新外设服务完后，再为原外设服务。
 - (2) 若新外设级别低或与原外设同级，则**CPU**继续为原外设服务，为原外设服务完后，再为新外设服务。

§ 5-2 MCS-51中断系统

MCS-51提供了5个中断源，2个中断优先级。

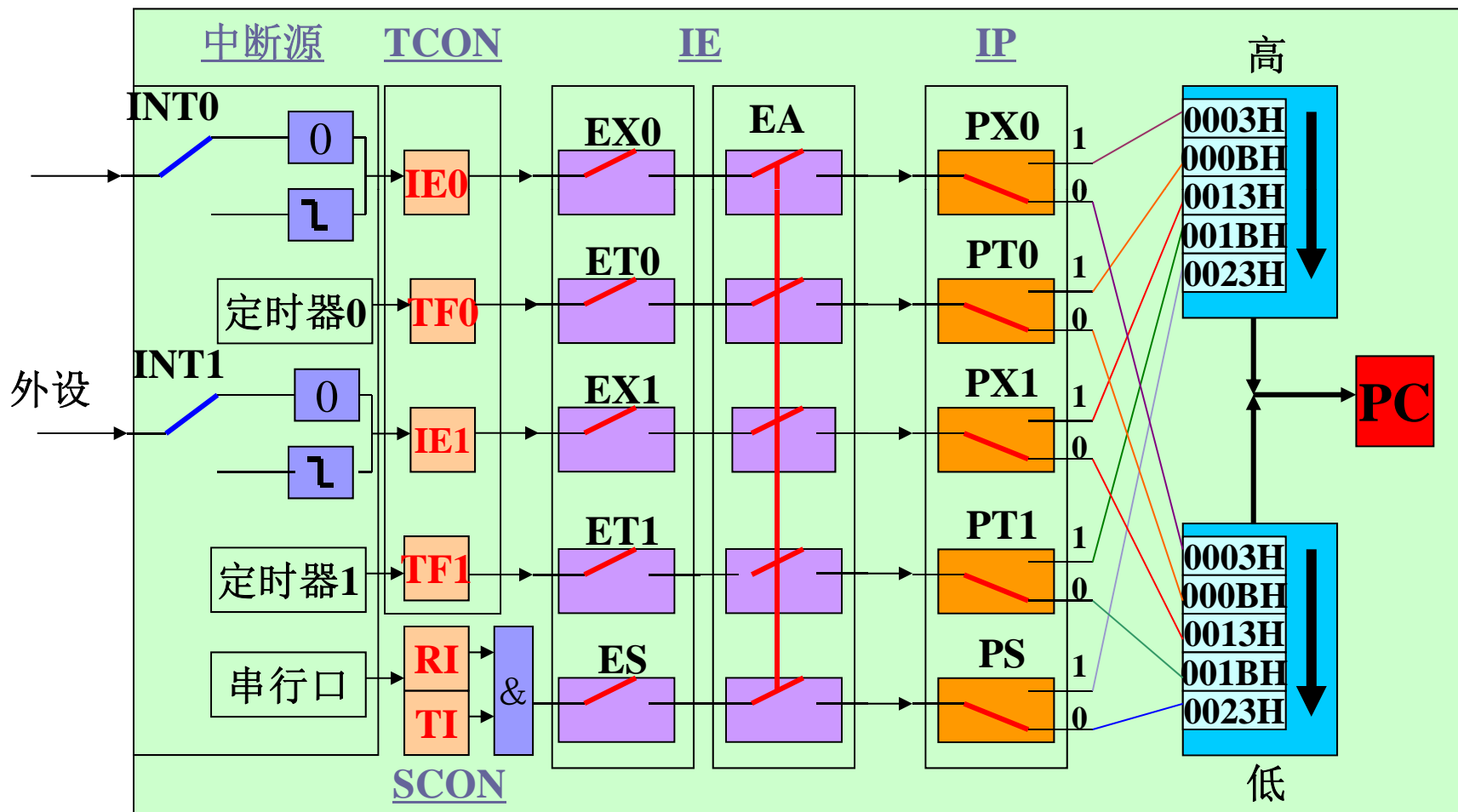
(1) 外部中断源（INT0和INT1）

- 由（ P3.2 ）端口线引入，低电平或下降沿引起。
- 由（ P3.3 ）端口线引入，低电平或下降沿引起。

(2) 内部中断源（T0、T1和TI/RI）

- T0：定时/计数器0中断，由T0回零溢出引起。
- T1：定时/计数器1中断，由T1回零溢出引起。
- TI/RI：串行I/O中断，完成一帧字符发送/接收引起。

MCS-51 中断系统结构图



中断相关的SFR:

- 中断源寄存器: TCON (88H) , SCON (98H)
- 中断允许控制寄存器: IE (A8H)
- 中断优先级控制寄存器: IP (B8H)
- 从使用者的角度出发, MCS51的中断系统就是一些SFR, 如IE、IP、TCON、SCON。

中断标志

- **CPU识别中断申请的依据:**

- CPU在每个机器周期的S6状态, 自动查询各个中断申请标志位, 若查到某标志位被置位, 将启动中断机制。
- MCS51单片机内部的**中断检测电路**检测到有申请后, 将检测结果存于TCON、SCON中;

每个中断源对应一个中断标志位, 当某个中断源有中断申请时, 相应的**中断标志位置1**, 各个中断源的中断标志位在TCON和SCON中。

- **TCON: 88H**

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----



可以按字节操作, 也可以按位操作。

- **SCON: 98H**

						TI	RI
--	--	--	--	--	--	----	----

可以按字节操作, 也可以按位操作。

中断标志（2）

- 在每条指令结束时，CPU检测各个中断标志位，若中断标志位置1，则认为有中断请求。
- 外中断有2种触发方式：低电平和下降沿，由TCON中的IT0和IT1决定。
 - (1) 当IT0=1，INT0为脉冲方式，后沿负跳变有效；当IT0=0，低电平有效。
 - (2) 当IT1=1，INT1为脉冲方式，后沿负跳变有效；当IT1=0，低电平有效。

中断标志（3）

- **TF0、TF1（Timer overflow interrupt flag）**定时器溢出中断标志
- 当定时器/计数器最高位进位时，置“1”**TF_i**表示正在向**CPU**申请中断，**CPU**响应中断后，自动清“0” **TF_i**。

串行口控制寄存器SCON (98H)

- SCON (98H)

						TI	RI
--	--	--	--	--	--	----	----
- TI/RI: 串行口发送/接收中断申请标志位
(由硬件自动置位, 必须由用户在中断服务程序中用软件清0)。
 - TI/RI=0: 没有串行口发送/接收中断申请;
 - TI/RI=1: 有串行口发送/接收中断申请。
- SCON的高6位用于串行口工作方式设置和串行口发送/接收控制。

中断开放与中断屏蔽控制寄存器 IE

- MCS-51单片机的5个中断源都是可屏蔽中断，也就是说用户可以通过软件方法来控制是否允许CPU去响应中断。
- 中断开放与 中断屏蔽是通过寄存器IE来控制。
- IE: A8H

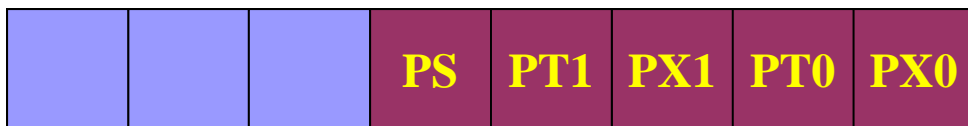
EA			ES	ET1	EX1	ET0	EX0
----	--	--	----	-----	-----	-----	-----
- IE可以按字节操作，也可以按位操作。

IE (2)

- **EA:** 当EA=0时, 称CPU关中断, 所有中断关闭。
当EA=1时, 称CPU开中断, 允许中断响应。
- **EX0:** 当EX0=0时, 禁止INT0 申请中断;
当EX0=1时, 允许INT0 申请中断。
- **EX1:** 当EX1=0时, 禁止INT1申请中断;
当EX1=1时, 允许INT1申请中断。
- **ET0:** 当ET0=0时, 禁止T0中断;
当ET0=1时, 允许T0中断。
- **ET1:** 当ET1=0时, 禁止T1中断;
当ET1=1时, 允许T1中断。
- **ES:** 当ES=0时, 禁止串行口中断;
当ES=1时, 允许串行口中断。
- **8051复位后IE=00H, 说明CPU和各个中断源都关中断。**

中断优先级寄存器 IP

- MCS-51单片机规定了两个中断优先级，对于每一个中断源均可编程为高优先级中断或低优先级中断，各中断源的优先级由中断优先级控制寄存器IP设定。
- IP: B8H 可以按字节操作，也可以按位操作。



- PX0: 当PX0=0时，INT0 处于低级；当PX0=1时，INT0 处于高级。
- PX1: 当PX1=0时，INT1 处于低级；当PX1=1时，INT1 处于高级。
- PT0: 当PT0=0时，T0 处于低级；当PT0=1时，T0 处于高级。
- PT1: 当PT1=0时，T1 处于低级；当PT1=1时，T1处于高级
- PS: 当PS=0时， 串行口处于低级；当PX0=1时，串行口处于高级。
- 8051复位后IP=00H，说明各个中断源都处于低级。

中断优先次序

8051单片机的中断优先级采用了自然优先级和人工设置高、低优先级的策略。

- 当五个中断源在同一个优先级时，就由自然优先级确定。INT0优先权最高，串行口优先权最低。在同一个优先级中，对五个中断源的优先次序安排如下：
INT0→T0→INT1→T1→串口（中断优先级从高到低）
- 开机时，每个中断都处于低优先级，中断优先级可以通过程序来设定，由中断优先级寄存器IP来统一管理。

例：中断设置

对寄存器IE、IP设置如下：

```
MOV IE, #10001111B
```

```
MOV IP, #00000110B
```

此时该系统中：

①CPU中断允许；

允许外部中断0、外部中断1、定时/计数器0、定时/计数器1发出的中断申请。

②允许中断源的中断优先次序为：

定时/计数器0 → 外部中断1 > 外部中断0 → 定时/计数器1。

中断处理过程（1）

1、中断请求

- (1) **MCS51**单片机内部的中断检测电路随时检测各个中断源，检测到有每个中断源有申请后，将相应的中断标志位置1。
- (2) **CPU**在每条指令结束时，检测各个中断标志位，若中断标志位置1，则认为有中断请求。
- (3) **CPU**读取**IE**和**IP**的内容，若中断允许且满足如下**条件**，则在下一个机器周期进入中断响应阶段。
 - ① 没有同级或更高级的中断正在执行（否则必须等**CPU**为它们服务完之后，才能响应新中断请求。）
 - ② 执行完的指令不是**RETI**或访问**IE**和**IP**的指令（否则必须另外执行一条指令后才能响应。）

中断处理过程（2）

2、中断响应

- 在中断响应阶段单片机做2件工作：
- (1) 断点地址压栈。
- (2) 根据不同的中断源，将不同的固定地址送**PC**，从而转到不同的地方执行程序。

- 各个中断源的入口地址是：

0003H、000BH、0013H、001BH、0023H。

INT0 T0 INT1 T1 串行口

- **注意：8051在响应了INT0、INT1、T0、T1的中断之后会自动清除它们的中断标志位，但不会清除串口的中断标志位。**

8051的5个中断源的中断服务入口地址之间相差8个单元。这8个存储单元用来存储中断服务程序一般来说是不够的。用户常在中断服务程序地址入口处放一条三字节的长转移指令。一般地，主程序从0030H单元以后开始存放。例如：

```
ORG 0000H
```

```
LJMP START      ; 转入主程序，START为主程序地址标号
```

```
ORG 0003H
```

```
LJMP INT0        ; 转外中断中断服务程序
```

```
ORG 000BH
```

```
LJMP T0          ; 转定时器T0中断服务程序
```

```
ORG 0030H
```

```
START: .....    ; 主程序开始
```

中断处理过程（3）

3、中断服务

- **CPU**响应中断后即转至中断服务程序的入口，执行中断服务程序。针对中断源的具体要求进行不同处理，不同的中断源其中断处理内容可能不同。
- 中断服务（子）程序的最后一条指令是**RETI**，**RETI**指令使程序返回被中断的（主）程序继续执行。
- **CPU**执行该指令，一方面清除中断响应时所置位的优先级有效触发器，释放对同级或低级中断申请的屏蔽；另一方面从堆栈栈顶弹出断点地址送入程序计数器**PC**，从而返回主程序。

中断响应时间

- 正常中断响应时间至少为**3~8**个机器周期，如果有同级或高级中断服务，将延长中断响应时间。
- 最短：查询中断标志（**1**）+**LCALL**（**2**）
- 最长：？？
正在执行**RETI**（**2**）+再执行一条指令（除法，**4**）+**LCALL**（**2**）

中断请求的清除（1）

- 为了避免中断请求标志没有及时清除而造成的重复响应同一中断请求的错误，**CPU**在响应中断时必须及时将其中断请求标志位清除。
- 8051的5个中断源的中断请求清除的方法是不同的。

（1）硬件自动清除

- 定时器溢出中断和下降沿触发的INTi得到响应后，其中断请求的标志位由硬件自动复位。

中断请求的清除 (2)

(2) 软件清除

串行口中断得到响应后，其中断请求的标志位TI和RI不能由硬件自动复位，必须由用户在中断服务程序的适当位置通过如下指令将它们清除。

CLR TI ;清除发送中断请求标志

CLR RI ;清除接收中断请求标志

或采用字节型指令：ANL SCON, #0FCH

中断请求的清除 (3)

(3) 外加硬件清除

对于电平触发方式的 INT_i ，引脚上的低电平须持续到中断发生。若中断返回前仍未及时撤除低电平，虽然CPU在响应中断时能由硬件自动复位 IE_0 或 IE_1 ，但引脚上的低电平仍会使已经复位的 IE_0 或 IE_1 再次置位，产生重复中断的错误。

§ 5-3 编写中断服务程序

中断入口、保护现场、关中断、中断服务主体程序、恢复现场、开中断、设置计数器、串行口的有关参数、中断返回指令 **RETI**。

中断服务程序初始化（1）

- 首先必须对中断系统进行初始化，包括：

1) 开中断，即设定**IE**寄存器。

□ **SETB EA** ； 开总中断控制位

□ **SETB EX0** ； 开外部中断0

□ **SETB ET0** ； 开定时器中断0

2) 设定中断优先级，即设置**IP**寄存器。

□ **SETB PT0**； 设定时器0中断为高优先级

中断服务程序初始化（2）

3) 如果是外部中断，还必须设定中断响应方式，即设定**IT0**、**IT1**位。

□ **SETB IT0** ; 设外部中断**0**为边沿触发方式

4) 如果是计数、定时中断必须先设定定时、计数的初始值。

□ **MOV TL0, #00H**

□ **MOV TH0, #4CH**

5) 初始化结束后，对于定时、计数器而言，还应该记得启动定时或计数，即设定**TR0**、**TR1**位。串口接收中断，要记得允许接收位**REN**应该设置。

□ **SETB TR0**

[例题1] 设8051外部中断源接引脚 INT0，中断触发方式为电平触发，试编制8051中断系统的初始化程序。

解：采用位操作指令实现（也可以采用传送指令和逻辑指令）。

SETB EA	; 开总中断
SETB EX0	; 开中断
SETB PX0	; 设置为高优先级
CLR IT0	; 设置为电平触发方式

编写中断服务程序(1)

- 中断服务程序，第一条指令必须安排在相应的中断入口地址，并且应该是转移指令，由于中断响应时，已经由硬件执行了 **LCALL** 指令，中断程序断点地址已经入栈，所以不能再用于子程序调用指令。
 - **ORG 0003H** ; 外部中断0入口地址
 - **LJMP INT_0**
 - **ORG 001BH** ; 定时器T0中断入口地址
 - **LJMP DELAY**

编写中断服务程序(2)

- 由于中断的产生是随机的，所以对中断程序中使用到的单元，必须在中断服务程序开始处，采用堆栈进行保护，即入栈。子程序返回前再出栈。

参数传递除外

- ☐ **PUSH ACC**

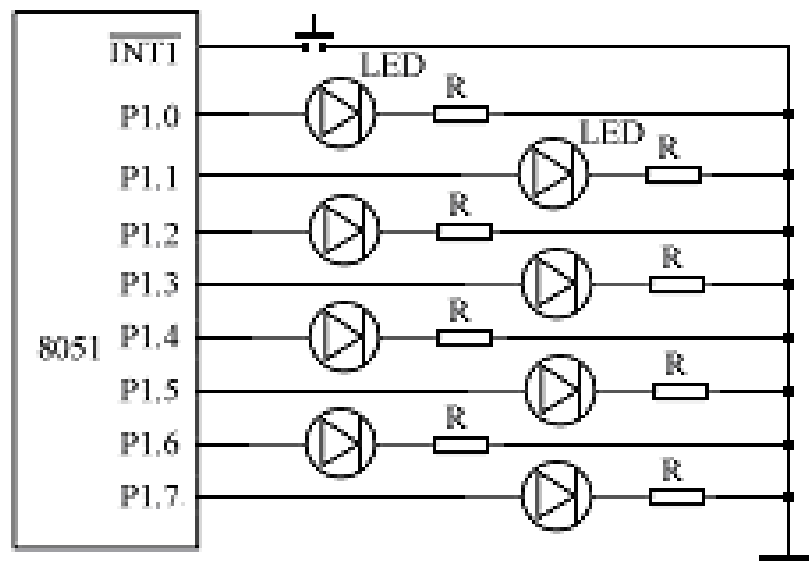
- ☐

- ☐ **POP ACC**

- 中断服务程序必须以**RETI**结束

[例题2] 通过外部中断控制八盏灯循环点亮。

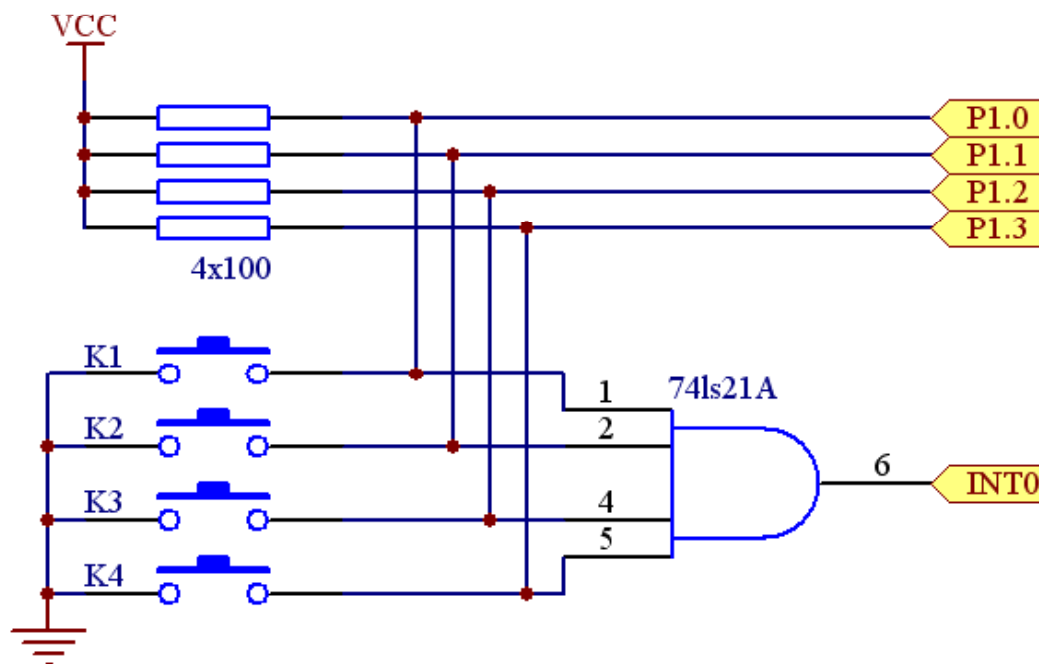
解：通过P1口扩展八盏灯，在INT1引脚接一个按钮开关到地，每按一下按钮就申请一次中断，点亮一盏灯，中断服务则是：依次点亮八盏灯中的一盏。采用边沿触发。硬件电路如下图所示。



程序如下：

```
ORG 0000H
LJMP MAIN
ORG 0013H      ; 中断服务程序入口地址
LJMP IN11
MAIN : SETB EA      ; 开总中断允许 “开关”
      SETB EX1      ; 开分中断允许 “开关”
      CLR  PX1      ; 低 优先级 ( 也可不要此句 )
      SETB IT1      ; 边沿触发
      MOV  A , #01H ; 给累加器A赋初值
      SJMP $        ; 原地等待中断申请
IN11 : RL  A         ; 左环移一次
      MOV  P1 , A    ; 输出到P1口
      RETI          ; 中断返回
END
```

[例题3] 查询方式扩展外部中断请求输入口的电路实现：



74ls21的逻辑表是：当输入全高时输出为高，任一输入低则输出就为低。
电路分析：当没有按键按下时，由于上拉电阻的作用，74ls21输出高，使INT0口为高电平，此时中断程序不被触发。如果任何一个按键按下，则74ls21输出低电平，触发中断程序。

中断程序设计为：如果程序设计检测次序是P1.0、P1.1、P1.2、P1.3口，优先级即按顺序排列，如果同时有多个键被按下，可根据实际要求设定为

- 1、仅仅只响应最优先中断；
- 2、按优先级顺序依次执行各个中断。

设计程序如下：

```
ORG    0000H
LJMP   MAIN
ORG    0003H           ; 外部中断0中断服务入口地址
LJMP   INT             ; 转中断服务
ORG    0100H
MAIN:  SETB  EA        ; 开总中断允许
       SETB  EX0       ; 开INT0中断
       SETB  IT0       ; 下降沿有效
       CLR   F0        ; 按键标志F0=0
LOOP:  JNB   F0, LOOP   ; 查询F0是否变为1
       CLR   F0
       JNB   P1.0, K1   ; K1按下转K1处理程序
       JNB   P1.1, K2   ; K2按下转K2处理程序
       JNB   P1.2, K3   ; K3按下转K3处理程序
       JNB   P1.3, K4   ; K4按下转K4处理程序
       SJMP  LOOP
       .....

```




```
K1:      .....      ; K1处理程序
          SJMP  LOOP
K2:      .....      ; K2处理程序
          SJMP  LOOP
K3:      .....      ; K3处理程序
          SJMP  LOOP
K4:      .....      ; K4处理程序
          SJMP  LOOP
          .....

          ORG  1000H
INT:     SETB  F0      ; 有键按下F0=1
          RETI
```

C51中断服务函数

1.一般形式：

**函数类型 函数名（形式参数表） [interrupt n]
[using n]**

interrupt 为关键字，**n**是中断号，是常量，取值范围为0-31。编译器从 $8n+3$ 处产生中断向量

using 为关键字，专门用来选择8051单片机中不同的工作寄存器组。**n**取值范围为0-3，是常量。

2. 关键字**using**对函数目标代码的影响如下：

如果不使用**using n**选项，在函数的入口处将当前工作寄存器组保护到堆栈中；函数返回之前将被保护的工作寄存器组从堆栈中恢复。

如果使用**using n**选项，编译器不产生保护和恢复代码。

在函数中确定一个工作寄存器组时必须十分小心，要保证任何寄存器组的切换都只在控制的区域内发生。

3. 关键字 `interrupt` 也不允许用于外部函数，它对中断函数目标代码的影响如下：

在进入中断函数时，特殊功能寄存器 ACC、B、DPH、DPL、PSW 将被保存入栈；如果不使用寄存器组切换，则将中断函数中所用到的全部工作寄存器都入栈；函数返回之前，所有的寄存器内容出栈。

4. 编写8051单片机中断函数应遵循以下规则：

- (1) 中断函数不能进行参数传递。
- (2) 中断函数没有返回值。
- (3) 在任何情况下都不能直接调用中断函数。
- (4) 如果在中断函数中调用了其他函数，则被调用函数所使用的寄存器组必须与中断函数相同。

void 函数名(void) interrupt n [using n]

```
#include <reg51.h>
unsigned char status;
bit flag;
void service_int() interrupt 0 using 2 //INT0中断函数，用第2组工作寄存器
{
    flag=1; //设置有键按下标志
    status=P1; //读入P1口状态
}
main()
{
    EA=1; EX0=1; IT0=1;
    while(1)
    {
        if(flag) //有键按下，则进行处理
        {
            switch(status) //根据P1口状态分支
            {
                case 7: ..... break; //K4按下(0111)处理
                case 11: ..... break; //K3按下(1011)处理
                case 13: ..... break; //K2按下(1101)处理
                case 14: ..... break; //K1按下(1110)处理
                default: ..... break; //无键按下情况处理
            }
            flag=0; //处理完，清标志
        }
    }
}
```

.....